Neural Networks.
Lab 5: Neural neworks with unsupervised competitive learning. Data clustering.

---

1. **Data clustering.** Let us consider a set of N-dimensional data which should be grouped in clusters such the data in the same cluster are similar and data in different cluster are dissimilar. The clusters labels are not known in advance (unlike the case of classification problems) thus the clusters are identified by a unsupervised learning process. One of the simplest (non-neural) clustering algorithm is the so-called kMeans algorithm based on an iterative application of two main steps: (i) assign the data to different clusters based on their distance to the clusters centers (the nearest neighbour criterion); (ii) recompute the centers of the clusters as averages of data in the cluster. Before the iterative process, the centers are randomly initialized with elements in the data set. The clustering problem can be solved by using a simple neural network trained by using an unsupervised learning algorithm based of a competitive type (for each training data one unit is selected to be a winner and only its weights are adjusted).

2. **Neural networks with unsupervised competitive learning.** The architecture of such a network is very simple: it consists of N input neurons (N is the dimension of data to be clustered) and M output units (M is the number of clusters to be identified). These two layers are fully connected in a feedforward way. Thus the connections weights are stored in a matrix W having M rows and N columns. For an input data X the answer of the network is the index of the winning unit. The winning unit is the output unit having the weights vector closest to X. If the distance between N dimensional vectors is denoted by d the criterion to decide that a unit i* is winning is:

$$-d(W^{i*}, X) + b_{i*} \geq -d(W^{i}, X) + b_i, \quad \text{for all } i \in \{1,...,M\}$$

The most used distance is the Euclidean one. The values bi are the biases used in Matlab to avoid the so-called "dead" units (units which do not become winners). The Matlab function used to define such a function is:

nc=newc([min1 max1; ...;minN maxN], M, lr, clr)

with the following significance of the parameters:

- min and max denote the minimum and the maximum of the values corresponding to the input unit i
- M denotes the number of clusters to be identified
- lr denotes the learning rate used in the adjustment ($W^{i*} = W^{i*} + lr(X - W^{i*})$) of the weights corresponding to the winning unit; it is an optional parameter and its default value is 0.01
- clr denotes the learning rate used to adjust the other units (in the case of "winner takes all" algorithm it should be 0); it is also an optional parameter having the default value 0.001.

Once created, the network can be trained by using the function train. The number of epochs (cycles through the data set) is set by using the corresponding field of the created object (nc.trainParam.epochs) . For a trained network one can decide to which cluster a new data

belongs by using the functions sim and vec2ind (in converts the result returned by sim in the index of the winning unit): result=vec2ind(sim(nc,data)).

3. Application 1: implementation of kMeans algorithm (the algorithm is described in Lecture 7)

```
function [omega]=kmeans(K)
% Parameters: K = the number of clusters
[x,n]=inputData(-1,1);     % read the input data (two dimensional
vectors with values in (-1,1))
[C]=initialization(x,n,K); % initializing the centers
d=[];
for t=1:50                 % the number of iterations is set to 50
    omega={};              % structure where the data are stored
    nomega=zeros(K);       % contains the number of data in each
cluster
% for each data the distances to all centers are computed
    for el=1:n
        for k=1:K
           d(k)=dist(x(:,el),C(:,k));
        end
% the closest center is identified
        kmin=1;
        for k=2:K
            if d(k)<d(kmin)
                kmin=k;
            end
        end
% the data is assigned to the cluster corresponding to the closest
center
        nomega(kmin)=nomega(kmin)+1;
        omega{kmin}(:,nomega(kmin))=x(:,el);
    end
% the new centers are computed
    for k=1:K
        disp(k);
        disp(omega{k});
        C(:,k)=average(omega{k});
    end
end
% the final classification of data in clusters
omega={};
nomega=zeros(K);
  for el=1:n
        for k=1:K
           d(k)=dist(x(:,el),C(:,k));
        end
        kmin=1;
        for k=2:K
            if d(k)<d(kmin)
                kmin=k;
            end
        end
        nomega(kmin)=nomega(kmin)+1;
        omega{kmin}(:,nomega(kmin))=x(:,el);
  end
```

```
% set the colors used to mark the points (at most 6 distinct clusters)
colors={'ro','go','bo','ko','yo','co'};
colors2={'r*','g*','b*','k*','y*','c*'};
for k=1:K
    plot(omega{k}(1,:),omega{k}(2,:),colors{k});
    plot(C(1,k),C(2,k),colors{k});
end
end
% function to get the coordinates of the points selected by the user
% inf = minimal value
% max = maximal value
% x = array 2xm containing the data (each column represents a data)
% n = the number of data
function [x,n]=inputData(inf,sup)
clf
axis([inf sup inf sup])
hold on
x = [];
n = 0;
b = 1;
while b == 1
[xi,yi,b] = ginput(1);
plot(xi,yi,'ro')
n = n+1;
x(1,n) = xi;
x(2,n) = yi;
end
end

% function to check if the initial centers are distinct
% ind = indices of already selected centers
% m = number of already selected centers
% i = index of the new center
function rez=is(ind,m,i)
rez=0;
for j=1:m
    if ind(j)==ind(i)
        rez=1;
    end
end
end



% function to initialize the centers of the clusters
% x = the input data
% n = the number of input data
% K = the number of clusters
function [C]=initialization(x,n,K)
ind=[];
for i=1:K
    ind(i)=random('Discrete Uniform',n);
    % check if the center was not selected before
    while is(ind,i-1,i)==1
        ind(i)=random('Discrete Uniform',n);
    end
end
```

```
for i=1:K
   C(:,i)=x(:,ind(i));
end
end
% function to compute the Euclidean distance between two vectors x and
% y
function d=dist(x,y)
n=length(x);
d=0;
for i=1:n
    d=d+(x(i)-y(i))^2;
end
end
% function to compute the centers of all clusters (as average of data %
% in a cluster)
function med=average(omega)
dim=size(omega);
med=[];
med=zeros(dim(1),1);
for i=1:dim(2)
    med=med+omega(:,i);
end
for i=1:dim(1)
    med(i,1)=med(i,1)/dim(2);
end
end
```

4. Application 2: implementation of a neural network trained with the "winner takes all" algorithm (WTA)

```
%  Data clustering with a competitive neural network
%  Parameters:
%     nrClusters: number of clusters
%     normalization:  0 (the data are not normalized); 1 (the data
should be normalized)
%     epochs: number of epochs
function [in,rca]=clusteringWTA(nrClusters,normalization,epochs)
in=[]; test=[];
clf
axis([-1 1 -1 1])
hold on
n = 0;
b = 1;
while b == 1
[xi,yi,b] = ginput(1);
plot(xi,yi,'ro')
n = n+1;
in(1,n) = xi;
in(2,n) = yi;
end

if normalization==1
 for i=1:size(in,2)
    in(:,i)=in(:,i)/sqrt(in(1,i)*in(1,i)+in(2,i)*in(2,i));
```

```
 end
 clf;
 plot(in(:,1),in(:,2),'ro');
end

minim1=min(in(1,:)); minim2=min(in(2,:));
maxim1=max(in(1,:)); maxim2=max(in(2,:));

nc=newc([minim1 maxim1; minim2 maxim2],nrClusters);     % network
creation
disp('The network was created. The training process starts ...');
nc.trainParam.epochs=epochs;
nct=train(nc,in);   % training
disp('The data are assigned to clusters ...');
clusterSize=[];
for k=1:nrClusters
    clusterSize(k)=1;
end
for i=1:size(in,2);
   rez=vec2ind(sim(nct,in(:,i)));
      cluster{rez}(:,clusterSize(rez))=in(:,i);
      clusterSize(rez)=clusterSize(rez)+1;
end
colors={'r*','b*','k*','y*','g*','c*'};
colorCenter={'r+','b+','k+','y+','g+','c+'};
% The clusters and their centers are plotted using different colors
for k=1:nrClusters
  for i=1:size(cluster{k},2)
    plot(cluster{k}(1,i),cluster{k}(2,i),colors{k});
    hold on
  end
  % the center of cluster k is plotted
  plot(nct.IW{1,1}(k,1),nct.IW{1,1}(k,2),colorCenter{k});
  hold on
end
disp('After visualizing (in the graphics window) the clusters press a
key ...')
pause

% Generating test data
test=2*rand(2,50)-1;
if normalization==1
  for i=1:size(test,2)
    test(:,i)=test(:,i)/sqrt(test(1,i)*test(1,i)+test(2,i)*test(2,i));
  end
end
clusterTestSize=[];
for k=1:nrClusters
    clusterTestSize(k)=1;
end
clf;
for i=1:size(test,2)
   rez=vec2ind(sim(nct,test(:,i)));
   for k=1:nrClusters
     if rez==k
      clusterTest{k}(:,clusterTestSize(k))=test(:,i);
```

```
      clusterTestSize(k)=clusterTestSize(k)+1;
     end
   end
end
for k=1:nrClusters
 for i=1:size(clusterTest{k},2)
  plot(clusterTest{k}(1,i),clusterTest{k}(2,i),colors{k});
  hold on;
 end
end
hold off
```

Exercise. Analyze the results obtained for the same set of data in the following situations:

- For different number of clusters#
- Without and with normalization
- For different values of the number of epochs